**Documentatie tema 5**

**Ghitun Patricia Roxana**

**Grupa 30227**

**1.Obiectivul temei**

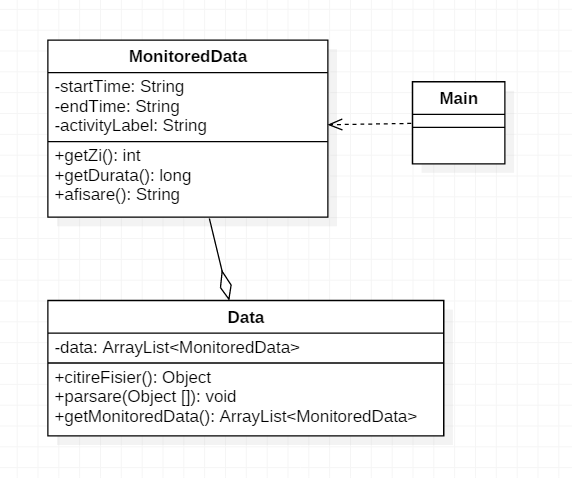
Obiectivul acestei teme este de a intelege si a invata cum se lucreaza cu stream-uri (procesul de stream processing ) si cu expresii Lambda(lambda expressions) , dar si de a ne invata principalele concepte si functionalitati aduse de Java 8 cum ar fi expresiile lambda si streamurile care ne prezinta metode mult mai usoare de a filtra date existente intr-o colectie de date, de a le modifica cu ajutorul unei functii , de a le numara unele proprietati sau de a le grupa in functie de un criteriu dorit.

**2. Analiza**

In aceasta aplicatie utilizatorul are un rol de supraveghetor sau de interpretator al datelor procesate de catre aplicatie . Acesta nu trebuie sa interactioneze deloc cu aplicatia printr-o interfata deoarece toate rezultatele vor fi afisate in consola si in fisiere text . Singurul rol pe care il detine utilizatorul este de a interpreta informatiile rezultate din functiile aplicatiei .

3. Proiectare

DIAGRAMA UML



Aplicatia contine trei clase : MonitoredData , Data si Main .

4. Implementare

Clasa **MonitoredData** are ca atribute start time – timpul de incepere (string) , end time – timpul de terminare (string) si activity label (String) – denumirea activitatii curente.

Aceasta clasa implementeaza metodele de get si set pentru aceste atribute pentru a permite altor clase sa acceseze sau sa modifice valorile atributelor obiectelor de tip Monitored Data . Pe langa aceste metode , in aceasta clasa mai sunt si metode precum : afisare (returneaza un string cu toate datele legate de un obiect de acest tip ) , getZi( metoda ce returneaza un intreg ce reprezinta ziua din string-ul de start time , si o metoda de getDurata ce returneaza o variabila de tip long ce reprezinta durata unei activitati ( diferenta dintre end time si start time ) calculata in milisecunde .

Clasa **Data** a fost implementata cu scopul citirii listei de activitati din fisierul dat in cerinta . Aceasta clasa are ca atribute fisierul si o lista de obiecte de tip MonitoredData luate din acel fisier .

Metodele implementate in aceasta clasa sunt cele de citireFisier , parsare si getMonitored Data .

* Metoda citire Fisier : aceasta metoda returneaza un sir de obiecte (stream-uri)
* Metoda parsare : aceasta metoda primeste ca parametru un sir de obiecte ( ulterior va fi apelata cu obiectede citite pentru a le parsa ). Pentru fiecare obiect din sirul respectiv , se va imparti folosind functia split in asa fel incat sa se poata separa stringurile de start time , end time si activity label . In urma parsarii , se creeaza un nou obiect de tipul Monitored Data ce va avea ca variabile instanta datele din urma parsarii . In final , noul obiect este adaugat in lista de obiecte de tipul Monitored Data ce este data ca atribut in aceasta clasa .
* Metoda get Monitored Data : aceasta metoda va returna un obiect de tip lista de obiecte Monitored Data in urma apelarii metodelor de citire fisier si parsare fisier . Metoda returneaza atributul clasei Data .
* Metoda afisare : in plus , este implementata si o metoda de afisare a datelor despre obiectele din aceasta lista de obiecte de tip monitored data .

Clasa **Main :** Aceasta clasa reprezinta clasa de baza a programului deoarece in ea se implementeaza toate operatiile cerute in cerinta problemei . Clasa aceasta nu are niciun atribut ci doar implementeaza metode corespunzatoare cerintelor din barem . Toate metodele din aceasta clasa sunt declarate statice .

* **Pentru realizarea primei cerinte**, se utilizeaza metoda task1() ( – afisarea numarului de zile distincte din fisier ) din aceasta clasa . In clasa MonitoredData a fost creata o metoda numita get ZI() ce returneaza o valoare de tip intreg ce returneaza ce returneaza ziua de inceput identica cu cea a obiectului initial . Aceasta metoda este de tip void deoarece in urma gasirii numarului de zile distincte , acesta este afisat in consola . Numarul de zile distincte este preluat cu ajutorul procesarii de streamuri . Initial se mapeaza ziua unui obiect d din lista de obiecte Monitored Data dupa care se apeleaza metoda distinct() pentru a le alege pe cele distincte si dupa care se apeleaza metoda count() pentru a numara cate aparitii sunt disticnte . Astfel , la final se va face un cast la tipul int pentru a avea numarul ca tip intreg .
* Pentru realizarea celei de a doua cerinte , a fost implementata metoda task2(Array List < Monitored Data > data ) . Aceasta metoda returneaza un HashMap ce contine ca si cheie un string – numele activitatii respectiva , si un intreg ce reprezinta numarul de aparitii a activitatii in sir . Pentru a numara aparitiile au fost folosite metodele computeIfPresent and putIfAbsent special implementate pentru streamuri .
  + **computeIfPresent()**:
    - default V computeIfPresent(K key, BiFunction<? super K,? super V,? extends V> remappingFunction)
    - Aceasta metoda are urmatoarea semnificatie : daca valoare specificata pentru Key este prezenta , deci apare , si nu este nula , atunci se va crea un nou loc in map in care va fi stocata noua cheie . In mapa noua se va stoca noua cheie si valoarea ei (value) .

In aceasta aplicatie , aceasta metoda a fost folosita cu scopul de a crea o mapa in care pe post de cheie este activitatea din fisier si pe post de valoare este numarul sau de aparitii in acel fisier .

* **putIfAbsent() :**
* **default V putIfAbsent(K key, V value)**
* Daca cheia specificata nu este deja asociata cu o valoare ( sau mapata cu un null ) atunci i se va asocia valoarea data si se va returna nul , altfel i se va returna valoarea curenta .
* In aplicatie , a fost folosita cu scopul de a pune in map o noua activitate cu numarul de aparitii initial setat la 1 . Adica , daca in momentul parcurgerii stream-ului , se gaseste o activitate ce nu a fost deja adaugata in map atunci aceasta va fi adaugata cu valoarea 1 . Asta are ca efect practic initializarea mapului cu activitatile .

Dupa ce a fost realizata mapa cu activitatile si numarul lor de aparitii , aceasta va fi returnata . Aceasta metoda este necesara pentru a scrie ulterior datele intr-un fisier „task2.txt” .

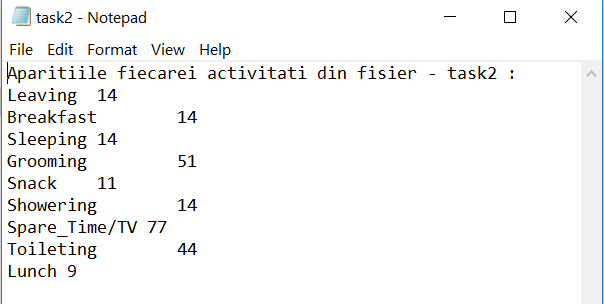
* **Pentru a treia cerinta din barem – task3()**  : A fost implementata metoda task3() . Similar cu cea de a doua cerinta , se creeaza un map ce contine ca si cheie un intreg ce reprezinta ziua si ca si valoare un alt map ce va reprezenta activitatea si numarul de aparitii al acesteia .
* **Pentru a patra cerinta – task4() :** Rezultatul pentru aceasta cerinta este o lista de valori intregi( un arraylist ) . Fiecare valoare intreaga corespunde unei linii din fisier si reprezinta durata activitatii descrise la linia respectiva. Pentru obtinerea acestei liste a fost implementata metoda getDurata() ce returneaza cate milisecunde sunt de la timpul de inceput la timpul de sfarsire . Pentru realizarea liste se folosesc metodele de groupingBy activity label si summlingLong get durata . Dupa ce a fost creata lista , in metoda de scriere se va pune in fisier doar activitatile ce au durata mai mare de 10 ore .
* **Pentru ultima cerinta –** task5() : A fost implementata metoda task5() . Pentru realizarea acestei cerinte a fost nevoie de trei liste ( doua map-uri si o lista de string-uri ) . Metoda returneaza o lista de activitati . Initial , a fost creata o lista de aparitii a tututor activitatilor , mapa initializata cu rezultatul celei de a doua cerinte . Dupa aceea , a fost creat un map ce este format din aparitiile activitatilor filtrate in functie de durata lor ( sa fie mai mica decat cinci minute < 300000 milisecunde ) . Dupa ce au fost create acest doua map-uri , datele din map-ul activitatilor cu durata mai mica de cinci minute a fost parcurs si daca numarul de aparitii este mai mare de 90% din numarul total de aparitii , atunci activitatea curenta este adaugata in lista de siruri de activitati . In final , se foloseste metoda de scriere in fisier task 5 pentru a scrie activitatile rezultate .

5. Rezultate

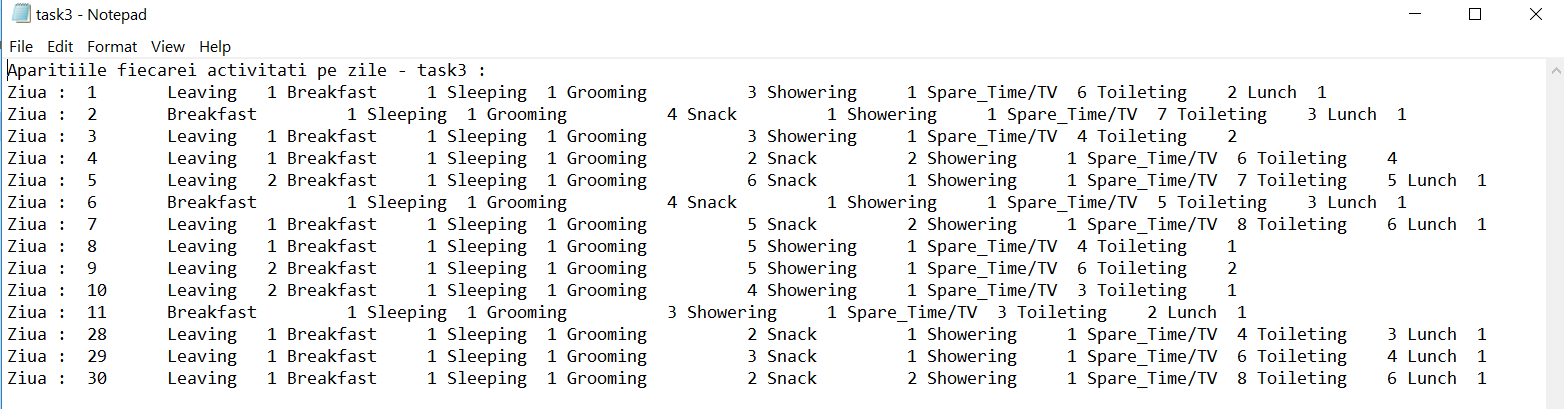
In urma rularii aplicatiei , fisierele corespunzatoare cerintelor vor contine rezultatele acestora.

Rezultatul pentru prima cerinta este afisat in consola : 14 activitati distincte .

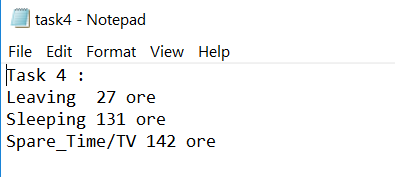
Rezultatul pentru a doua cerinta este scrie in fisierul task2 .txt : Acesta contine, incepand cu linia a doua, numele fiecarei activitati urmate de numarul de aparitii al acesteia pe parcursul perioadei de monitorizare.



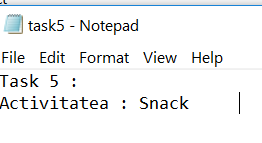
Rezultatele pentru a treia cerinta sunt scrise in fisierul task3 .txt . Incepand cu a doua linie, in fisier va fi afisata ziua, urmata de activitatile effectuate in acea zi alaturi de numarul lor de aparitii.



Rezultatele pentru a patra cerinta sunt scrise in fisierul txt task4 . Incepand cu linia a doua sunt afisare activitatile ce au o durata mai mare de zece ore pe parcursul intregii monitorizari .



Pentru cea de- a cincea cerinta rezultatul a fost afisate in fisierul task5.txt. In fisier au fost scrise doar activitatile ce au un numar de aparitii cu o durata mai mica de cinci minute mai mare de 90 la suta din numarul total de aparitii .



6. Concluzii

In urma realizarii acestei teme au fost invatate si consolidate mai multe concepte , cum ar fi ce sunt stream-urile si cum se folosesc, ce sunt expresiile lambda si cum usureaza acestea munca, reducand numarul de linii de cod ai aplicatiei .

O functie lambda este o functie definita si apelata fara a fi legata de un identificator. Functiile lambda sunt o forma de functii ,,incuibate” (nested functions) in sensul ca permit accesul la variabilele din domeniul functiei in care sunt continute. Functiile anonime au fost introduse de catre Alonzo Church in anul 1936, in teoria sa despre calculele lambda.

One issue with anonymous classes is that if the implementation of your anonymous class is very simple, such as an interface that contains only one method, then the syntax of anonymous classes may seem unwieldy and unclear. In these cases, you're usually trying to pass functionality as an argument to another method, such as what action should be taken when someone clicks a button. Lambda expressions enable you to do this, to treat functionality as method argument, or code as data.

Expresiile Lambda ne permit sa cream instanae ale claselor cu o singura metoda intr-un mod mult mai compact. O expresie lambda consta:

• dintr-o lista de parametri formali, separati prin virgula si cuprinsi eventual intre paranteze rotunde, • sageata directionala ->,

• un body ce consta dintr-o expresie sau un bloc de instructiuni.

Stream este un concept nou introdus de Java 8. Folosind stream, putem accesa date in mod declarative asemanatoare statement-urilor SQL. Stream-ul reprezinta o succesiune de obiecte care suporta operatii aggregate. Caracteristicile unui stream sunt:

- Succesiune de elemente: un stream furnizeaza o multime de elemente de un anumit tip. Un stream nu stocheaza niciodata elementele

- Sursa: un stream primeste ca intrare obiecte de tip Collection, Array sau dispositive de intrare/iesire

- Operatii agregate: un stream suporta operatiile agregate, cum ar fi filter, map, limit, find, match

- Pipelining: multe dintre operatiile efectuate de un stream returneaza stream-ul. Aceste operatii se numesc opeartii intermediare

- Iteratii automate: operatiile unui stream efectueaza iteratiile intern asupra elementelor, spre deosebire de Collections care necesita iteratii explicite.

Astfel, conceptul de flux oferă posibilitatea tratării unitare a oricărui tip de comunicație, cu orice tip de entitate din sistem. Definirea unui flux se face prin definirea capetelor lui (a sursei și a destinației). Datorită faptului că există două direcții de comunicare, se poate face o separare în două tipuri de fluxuri: flux de intrare (input stream) și flux de ieșire (output stream).

În Java, toate fluxurile standard sunt accesate prin clasa java.lang.System. Astfel, pentru Standard Input există fluxul System.in, pentru Standard Output, System.out iar pentru Standard Error, System.err. Cei trei membri in, out, err ai clasei System sunt statici.

7. Bibliografie

<https://docs.oracle.com/javase/8/docs/api/java/util/stream/package-summary.html>

<https://www.tutorialspoint.com/java8/java8_streams.htm>

<http://winterbe.com/posts/2014/07/31/java8-stream-tutorial-examples/>

<https://stackoverflow.com/questions/24872531/concurrenthashmap-jdk-8-when-to-use-computeifpresent> - compute if present

<http://www.topjavatutorial.com/java-8/java-8-map-computeifpresent/>

<http://javatutorialhq.com/java/util/hashmap-class/putifabsent-method-example/>

<https://www.mkyong.com/java/how-to-calculate-date-time-difference-in-java/>

<http://www.baeldung.com/java-groupingby-collector> - grouping by TASK4

<https://www.logicbig.com/how-to/code-snippets/jcode-java-8-streams-collectors-summinglong.html> - summingLong – task 4

<https://www.mkyong.com/java/how-to-write-to-file-in-java-bufferedwriter-example/> - write in fisier cu buffered writer